INFORMATION AND COMPUTATION 92, 81–96 (1991)

Deterministic P-RAM Simulation with Constant Redundancy*

SCOT W. HORNICK

Andersen Consulting, Center for Strategic Technology Research, 100 S. Wacker Dr., Chicago, Illinois 60606

AND

FRANCO P. PREPARATA

Department of Computer Science, Brown University, Box 1910, Providence, Rhode Island 02912

In this paper, we show that distributing the memory of a parallel computer and, thereby, decreasing its granularity allows a reduction in the redundancy required to achieve polylog simulation time for each P-RAM step. Previously, realistic models of parallel computation assigned one memory module to each processor and, as a result, insisted on relatively coarse-grain memory. We propose, on the other hand, a more flexible, but equally valid model of computation, the *distributed-memory, bounded-degree network* (DMBDN) model. This model allows the use of fine-grain memory while maintaining the realism of a bounded-degree interconnection network. We describe a P-RAM simulation scheme, which is admitted under the DMBDN model, that exploits the increased memory bandwidth provided by a two-dimensional mesh of trees (2DMOT) network to achieve an overhead in memory redundancy lower than that required by other fast, deterministic P-RAM simulations. Specifically, for a deterministic simulation of an n-processor P-RAM on a bounded-degree network, we are able to reduce the number of copies of each variable from $O(\log n/\log\log n)$ to $\Theta(1)$ and still simulate each P-RAM step in polylog time.

1. INTRODUCTION

Considerable research has been devoted to developing general-purpose architectures that exploit the parallelism offered by modem integration technology. A popular theoretical approach to this problem has been the design of processor networks for the simulation of abstract models of com-

* This research was supported in part by the Semiconductor Research Corporation under contract 87-DP-109.

0890-5401/91 $3.00

putation, such as the *parallel, random-access machine* (P-RAM) model (Upfal, 1984; Upfal and Wigderson, 1987; Mehlhorn and Vishkin, 1984; Karlin and Upfal, 1986; Alt *et al.*, 1987; Ranade, 1988; Luccio *et al.*, 1988, 1990). The P-RAM model of computation, formalized by Fortune and Wyllie (1978) and used even earlier by Hirschberg (1977) and Preparata (1977), has been a valuable tool for theoretical computer scientists studying the power and fundamental limitations of parallelism (see (Karp and Ramachandran, 1988) for a survey of results). By assuming the existence of a shared memory accessible to all processors in $O(1)$ time, the P-RAM model trivializes the problem of inter-processor communication to reveal the inherent parallelism in a problem and facilitate the development of parallel algorithms.

Formally, a P-RAM consists of n sequential processors (RAMs) and m shared memory cells (Fig. 1). These processors operate synchronously and, at each step, each one fetches an instruction from a private RAM and executes it. Executing these instructions may require accesses to the shared memory, and, in particular, the processors may all simultaneously read from or write to the shared memory at any given step. Several variants of the P-RAM have been defined, each differing in the convention applied to handle read/write conflicts, i.e., attempts by more than one processor to access the same memory cell in the same step. P-RAMs are either *exclusive-read* (ER) or *concurrent-read* (CR) and either *exclusive-write* (EW) or *concurrent-write* (CW). The most restrictive of these is the EREW P-RAM in which no memory cell may be accessed by more than one processor in a given step. The least restrictive is the CRCW P-RAM, in which simultaneous reading and writing of memory cells is allowed, with some rule defining the exact semantics of simultaneous writes.

Of course, the P-RAM model is not technologically feasible for a large number of processors. Therefore, research has been directed toward simulating the P-RAM model on more realistic models of computation, models that account for communication costs. The two most common among these are the *module parallel computer* (MPC) model (Mehlhorn and Vishkin, 1984) and the *bounded-degree network* (BDN) model (Alt *et al.*, 1987). The MPC model takes into account the fact that it is not feasible

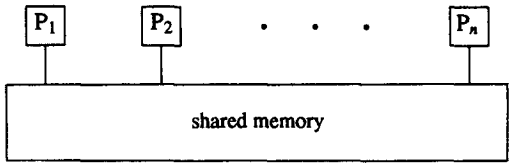


FIG. 1. The P-RAM model of computation.

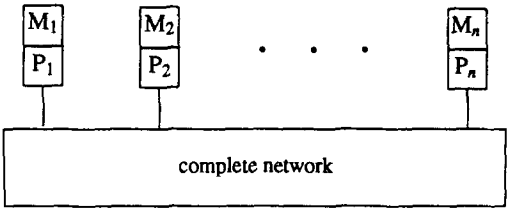


FIG. 2. The MPC model of computation.

for any more than a constant number of processors to simultaneously access the same memory module. It consists of n RAM processors, each equipped with a memory module containing m/n memory cells, and all interconnected by the complete graph K_n (Fig. 2). However, this model itself is not feasible because the complete graph interconnecting the processors cannot be realized without unbounded fan-in or fan-out. This led to the consideration of the BDN model, in which each processor is linked directly to only a constant number of other processors (Fig. 3).

Mehlhorn and Vishkin (1984) showed that the MPC model can *probabilistic* simulate T steps of a P-RAM in $O(T \log n)$ time by using universal hashing and by increasing the capacity of each memory module to $O(m/n \log n)$. Upfal (1984) proved a similar result, and gave an $O(T \log^2 n)$ time probabilistic simulation on a BDN. This result was subsequently improved by Karlin and Upfal (1986), who described a $\Theta(T \log n)$ time probabilistic simulation on a BDN (which is optimal with respect to time), and by Ranade (1987), who reduced the size of the queues used in the simulation from $O(\log n)$ to $\Theta(1)$.

The first reasonable *deterministic* P-RAM simulation on an MPC was that of Upfal and Wigderson (1987), which uses $O(\log n)$ time-stamped copies of each variable to simulate each P-RAM step in $O(\log n(\log \log n)^2)$ (assuming m is polynomial in n). Alt *et al.* (1987) improved this upper bound to $O(\log m)$ time and used this simulation along with a sorting network to give an $O(\log n \log m)$ time simulation on a BDN. They also proved a lower bound of $\Omega(\min\{\sqrt{n \log n},$

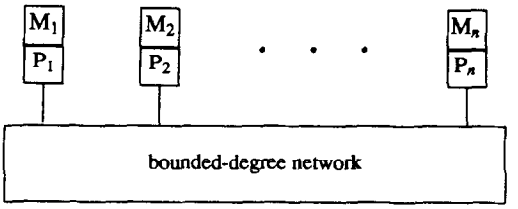


FIG. 3. The BDN model of computation.

$\log n \log m/\log\log m\}$) on the time required to simulate a P-RAM step on a BDN if all communication is required to be *point-to-point*, i.e., if a processor has to send a separate message to update each copy of a variable. (The same result was obtained independently in (Karlin and Upfal, 1986).) Recently, Herley and Bilardi (1988) achieved this time lower bound and reduced the *redundancy* r, i.e., the required number of copies of each variable, to $r = \Theta(\log m/\log\log m)$ by using bounded-degree networks based on certain expander graphs.

Luccio *et al.* have recently suggested the *two-dimensional mesh of trees* (2DMOT) as a practical bounded-degree network for the simulation of P-RAMS with m polynomial in n. In (Luccio *et al.*, 1988), they proposed this network for the probabilistic simulation of P-RAMs, and in (Luccio *et al.*, 1990) they proposed it for deterministic P-RAM simulation. The 2DMOT was originally proposed by Nath *et al.* (1983) (where it was referred to as the "orthogonal trees" network) as an appropriate VLSI architecture for computing matrix–vector products and for a variety of other related matrix and graph problems. For n a power of 2, an $(n \times n)$-2DMOT network consists of n^2 processors $P(i, j)$, for integral $i, j \in [1, n]$, and two families of n fully balanced binary trees connecting them as follows:

(1) Row trees: RT(i) with the processors $P(i, j)$ for $j \in [1, n]$ as their leaves, and

(2) Column trees: CT(j) with the processors $P(i, j)$ for $i \in [1, n]$ as their leaves (Fig. 4).

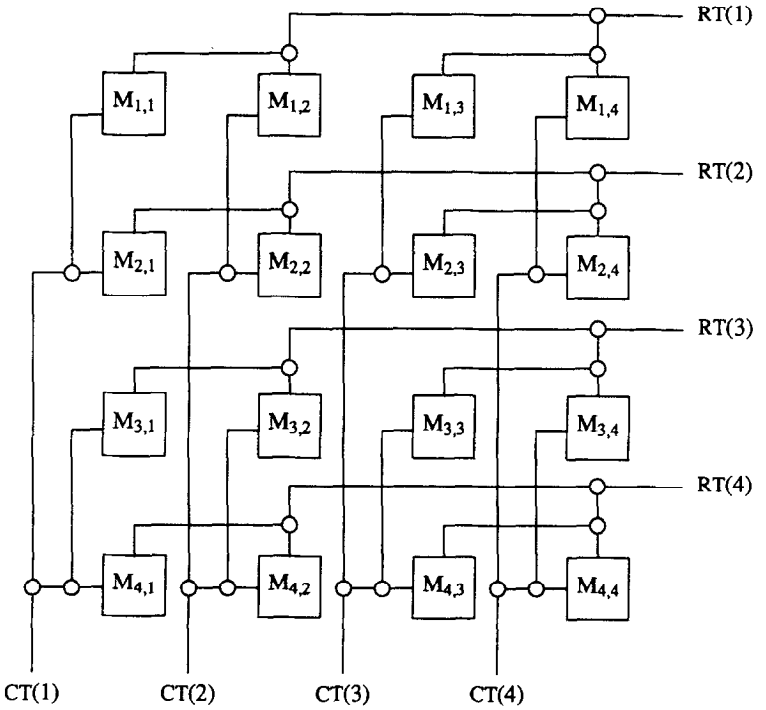
The VLSI area occupied by obvious layouts of the $(n \times n)$-2DMOT (e.g., that in Fig. 4) is $\Theta(n^2(\log^2 n + A_l))$, where A_l is the area of a leaf processor (assuming that leaf processors are as large or larger than the processors within the trees). Leighton (1984), who coined the term "mesh of trees," proved the optimality of this upper bound.

Since our simulation scheme depends on the results of (Luccio *et al.*, 1990), which in turn depend on the results in (Upfal and Wigderson, 1987), it is worthwhile to review those results in a little more detail. In their deterministic MPC simulation of a P-RAM, Upfal and Wigderson adopted a strategy, sometimes referred to as *majority rule*, first proposed by Thomas (1979) and Gifford (1979) in the context of distributed database theory. Their scheme distributes $r = 2c - 1$ copies of each P-RAM variable u among the n memory modules of the MPC. Stored along with each copy is a time stamp indicating the time at which the copy was last written. The scheme simulates each P-RAM step in turn. When a step in which u is written is simulated, at least c copies of u are updated. When a step in which u is read is simulated, at least c copies of u are retrieved, with the correct value given by the copy having the most recent time stamp.

Because of the symmetry of the read and write operations, they can be thought of and described jointly as access operations. At any point during the simulation of a given P-RAM step, a variable being accessed in that step is referred to either *as live*, if fewer than c copies of the variable have been accessed thus far, or as *dead*, if c or more copies of the variable have already been accessed. T*he live copies* of a live variable are those that have not yet been accessed. Whenever a live variable "dies," no further attempts are made to retrieve those copies that remain unaccessed. In this way, dead variables cannot contend for memory.

The success of this scheme depends on a lemma showing that it is possible to map the copies of the variables to the processors such that, for any sufficiently small set of live variables, a significant fraction of the live copies reside in distinct processors.

LEMMA 1 (Upfal and Wigderson, 1987). *For constant $b > 4$ and n sufficiently large, there is a $c = O(\log m/\log b)$ such that there is a way to distribute the $2c-1$ copies of each variable among the processors and ensure that, for any set of $q \leqslant n/(2c-1)$ live variables, the live copies reside in at least $(2c-1)q/b$ distinct processors.*



This same memory map is employed in the deterministic simulation scheme of (Luccio *et al.*, 1990). The roots of the row and column trees of the 2DMOT are identified (i.e., coalesced), and the processors are located at the roots, with the rest of the 2DMOT acting as a switching network to route communication packets between the processors. As in (Upfal and Wigderson, 1987) the processors are organized into $n/(2c-1)$ *clusters* of $2c-1$ processors each. The simulation proceeds in two stages: In the first stage, the processors of a cluster cooperate to access the copies of the variable requested by each processor in the cluster in succession; each request is processed in $O(\log \log n)$ phases, with the requests for the $2c-1$ cluster variables interleaved in time. It is shown that the first stage leaves at most $n/(2c-1)$ unsatisfied requests (i.e., live variables). The second stage is devoted to accessing these variables. Again, the processors of a cluster cooperate in requesting the same variable in successive phases, but now there is only one variable per cluster, and the copy access requests are queued, with $O(\log n)$ requests satisfied per phase to match the $O(\log n)$ latency of the 2DMOT. It is shown that $O(\log n/\log \log n)$ phases suffice to complete this stage. Since each phase (of either stage) takes $O(\log n)$ time, their scheme simulates a P-RAM step in $O(\log^2 n/\log \log n)$ time.

This matches the time performance of (Herley and Bilardi, 1988) and is an improvement in the sense that the 2DMOT is not plagued by the large constants of constructive expander graphs. On the other hand, the (Luccio *et al.*, 1990) simulation has $O(\log n)$ redundancy, as opposed to the $\Theta(\log n/\log \log n)$ redundancy of (Herley and Bilardi, 1988), and the objection can be raised that it introduces additional processors (albeit mere switches) in the interconnection network. Indeed, this raises the question of how much can be gained by relaxing some of the restrictions of the BDN model.

The main contribution of this paper is the elucidation of the crucial role played by memory granularity on the redundancy required for deterministic P-RAM simulation. In particular, in Section 2 we will show that a variant of the MPC with n processors and M memory modules can simulate a P-RAM in polylog time with *constant redundancy* provided that $M = n^{1+\varepsilon}$, for $\varepsilon > 0$; here $\varepsilon > 0$ is the characteristic condition of fine granularity. Then, in Section 3, we propose a *distributed-memory, bounded-degree network* model of computation that allows the use of more memory modules and the introduction of switches in the interconnection network. We show how the 2DMOT architecture can be used in conjunction with fine-grain memories to obtain a fast deterministic P-RAM simulation scheme with constant redundancy. Our scheme places the processors at the roots of the trees, but, in contrast with that of (Luccio *et al.*, 1990), separates the memory cells from the processors and distributes them among the leaves of the 2DMOT. In a P-RAM with a large memory, this

exploits the 2DMOT in a much more powerful manner by increasing the bandwidth to the memory. As a result, by ensuring that the memory "granule" is not exceedingly small, the VLSI area occupied by the memory of the simulating network (excluding the memory map) is on the same order as that occupied by the memory of the P-RAM itself.

An alternative scheme to achieve constant redundancy has been recently proposed by Schuster (1987). This scheme uses the information dispersal-recovery method suggested by Rabin (1989), whereby a file of b elements of a finite field is recoded into a file of $d > b$ elements from the same field, with the property that any b of the elements of the latter permit the recovery of the original file. The shared memory is subdivided into m/b blocks of size b, and data are stored in recoded form (i.e., each stored block has size d). A variable belongs to a block; to access a variable it is sufficient to access $(d+b)/2$ terms of its block. By choosing b and d both $\Theta(\log n)$, memory size increases only by a constant factor, although as many as $\Theta(\log n)$ variables may have to be processed per variable accessed.

2. The Effect of Memory Granularity on the Redundancy of Deterministic P-RAM Simulation

We begin by noting that the MPC and BDN models impose limitations on the simulation which may or may not actually correspond to the economic/physical constraints of an implementation. In particular, in a P-RAM with a large memory, say $m = n^{2+\delta}$ for $\delta > 0$, these models force the module size to be very large, at least $m/n = n^{1+\delta}$. Since the only access to a module is through the associated processor, a great deal of contention can occur. In other words, we have essentially imported the "von Neumann bottleneck" from conventional serial computation to the P-RAM simulation problem. Now, in parallel systems that must be built by interconnecting several conventional von Neumann machines, this may be a reasonable constraint. However, in systems that can be built "from scratch," considerable advantage can be gained by distributing the memory as an entity separate from the processors.

This consideration motivates the definition of an alternative model, the *distributed-memory, module parallel computer* (DMMPC) model. In this model, the n processors are interconnected to $M = rm/g$ memory modules by the complete bipartite graph $K_{n,M}$ (Fig. 5). The quantity g is called the *granularity*, i.e., the number of memory cells in each module.

The original definition of the MPC model in (Mehlhorn and Vishkin, 1984) actually allows the flexibility of having more memory modules than processors, but subsequent usage (Alt *et al.*, 1987) restricted the model so that each memory module is associated with a unique processor, as

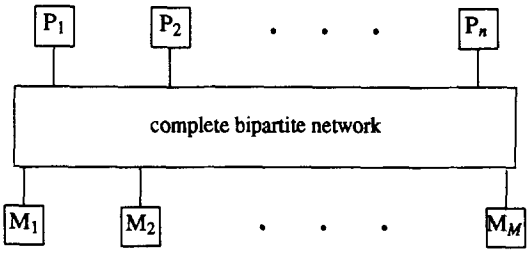


FIG. 5. The DMMPC model of computation.

described in Section 1. By distriminating between the MPC model and the DMMPC model, we will avoid any possible confusion.

In (Mehlhorn and Vishkin, 1984), the effect of memory granularity on probabilistic P-RAM simulations on a DMMPC was studied. Mehlhorn and Vishkin showed that increasing M simplified the class of hash functions required to ensure polylog expected-time performance. In this section, we undertake a similar study for deterministic P-RAM simulations on a DMMPC. We will show that increasing M reduces the redundancy required to ensure polylog worst-case-time performance.

Upfal and Wigderson (1987) proved that the time taken to simulate a P-RAM step by any MPC simulation scheme which updates an average of ρ copies of each variable is $\Omega((m/n)^{1/(2\rho)})$. In other words, the redundancy of a deterministic P-RAM simulation must be $\Omega(\log m/\log\log m)$ to ensure polylog time on an MPC, a bound later achieved by Herley and Bilardi (1988). Our claim may be somewhat surprising in view of this result; therefore, we prove here an analogous result for the DMMPC. The following theorem demonstrates the critical role played by memory granularity in establishing lower bounds on redundancy.

THEOREM 1. *Any P-RAM simulation scheme running on a DMMPC with n processors, $M = n^{1+\varepsilon}$ memory modules, and $m = n^k$ variables requires redundancy $r = \Omega((k-1)\log n/(\varepsilon \log n + \log h))$ to simulate an arbitrary P-RAM step in time h, where $h < o(n/\log m)$.*

Proof. Call S the collection of all $\binom{M}{n/h-1}$ possible sets of $n/h-1$ memory modules (assuming, for simplicity, that h divides n). No such set of memory modules contains all the updated copies of n variables; if one did, then a P-RAM step updating those variables would require simulation time $n/(n/h-1) > h$ (since each updated variable must have at least one updated copy), which is a contradiction. Thus, each member of S can contain all the updated copies of at most $n-1$ variables. The situation can be modeled by an $\binom{M}{n/h-1} \times m$ 0–1 matrix, the rows of which are indexed by

the sets of S, conventionally numbered from 1 to $\binom{M}{n/h-1}$, and the columns of which are indexed by the m variables. The (i, j) entry of the matrix is set equal to 1 if and only if all the updated copies of the jth variable reside in set number i of S. So, in each row of this matrix, there can be at most $n-1$ ones, which gives a maximum of $\binom{M}{n/h-1}(n-1)$ ones in the matrix.

Let ρ be the average number of copies of each variable that are updated. Clearly, $\rho \leqslant r$, and the number of variables with 2ρ or fewer updated copies is at least $m/2$. We wish to obtain a lower bound on the number of sets of S containing the $j \leqslant 2\rho$ updated copies of one such variable. This lower bound is $\binom{M-j}{n/h-1-j} \geqslant \binom{M-2\rho}{n/h-1-2\rho}$, and it is attained when each updated copy belongs to a distinct memory module. Thus, in each column of the matrix corresponding to the variables with 2ρ or fewer updated copies, there are at least $\binom{M-2\rho}{n/h-1-2\rho}$ ones. Therefore, the number of ones in the matrix is at least $(m/2)\binom{M-2\rho}{n/h-1-2\rho}$. Comparing this with the maximum number of ones gives

$$\binom{M}{n/h-1}(n-1) \geqslant \frac{m}{2}\binom{M-2\rho}{n/h-1-2\rho}$$

$$\frac{\binom{M}{n/h-1}}{\binom{M-2\rho}{n/h-1-2\rho}} \geqslant \frac{m}{2(n-1)},$$

which implies

$$\left(\frac{M-2\rho+1}{n/h-2\rho}\right)^{2\rho} > \frac{m}{2n}.$$

Manipulating this equation and taking the logarithm yields

$$\rho > \frac{\log m - \log n - 1}{2[\log(M-2\rho+1) - \log(n/h-2\rho)]},$$

which is satisfied by some

$$\rho = \Omega\left(\frac{\log m - \log n}{\log M - \log n + \log h}\right)$$

for $h < o(n/\log m)$. Since $r \geqslant \rho$, we obtain finally

$$r = \Omega\left(\frac{(k-1)\log n}{\varepsilon \log n + \log h}\right). \quad \blacksquare$$

When $k > 1$ and $\varepsilon > 0$ are constants and h is a polynomial in $\log n$, this generalization of Theorem 4.1 in (Upfal and Wigderson, 1987) only yields

a constant as a lower bound on the redundancy. Note that $k=1$ corresponds to the trivial case of one variable per processor (so that no contention arises) and $\varepsilon=0$ corresponds to one memory module per processor. This illustrates the crucial role played by granularity in achieving constant redundancy. Indeed, constant redundancy is achievable in this case, as we will now show.

The algorithm described in (Upfal and Wigderson, 1987) can be used with only a minor modification, namely, an improvement in the argument parameter c obtained by tightening Lemma 1 for the case $M=n^{1+\varepsilon}$.

LEMMA 2. *For constants $b>2$ and $c>(bk-\varepsilon)/(\varepsilon(b-2))$ and n sufficiently large, there is a way to distribute the $2c$-1 copies of each variable among the M memory modules such that, for any set of $q\leqslant n/(2c-1)$ live variables, the live copies occupy at least $(2c-1)q/b$ distinct modules.*

Proof. A memory map is "bad" if it does not satisfy the conditions of the theorem, i.e., if there exists some choice of $q\leqslant n/(2c-1)$ variables and some choice of c live copies of each of these variables such that the live copies occupy fewer than $(2c-1)q/b$ memory modules. We show that, asymptotically, the number of "bad" memory maps is a small fraction f of the total number of possible memory maps.

There are $\binom{m}{q}$ ways to choose the q live variables, and the number of ways in which a memory map can be "bad" for a particular set of q variables is less than

$$\binom{2c-1}{c}^q \binom{M}{(2c-1)q/b} \frac{[(2c-1)^2\, qm/(bM)]!}{[(2c-1)^2\, qm/(bM)-cq]!}\,[2c-1)m-cq]!.$$

The first factor is the number of ways to choose the live copies of the q live variables, the second is the number of ways to choose the set of congested memory modules, the third is the number of ways to map the live copies to the congested memory modules, and the last is the number of ways to map the remaining copies of all variables (live or dead). Applying the union bound and dividing by $[(2c-1)m]!$, the total number of possible memory maps, we obtain

$$\begin{aligned} f &< \sum_{q\leqslant n/(2c-1)} \binom{m}{q}\binom{2c-1}{c}^q \binom{M}{(2c-1)q/b} \\ &\quad \times \frac{[(2c-1)^2\, qm/(bM)]!}{[(2c-1)^2\, qm/(bM)-cq]!}\,\frac{[(2c-1)m-cq]!}{[(2c-1)m]!} \\ &< \sum_{q\leqslant n/(2c-1)} \binom{m}{q}\binom{2c-1}{c}^q \binom{M}{(2c-1)q/b}\left[\frac{(2c-1)q}{bM}\right]^{cq}. \end{aligned}$$

Using the inequalities $\binom{\alpha}{\beta} < (e\alpha/\beta)^{\beta}$ and $\binom{2c}{c} < 2^{2c}/\sqrt{c}$, we can write

$$f < \sum_{q \leq n/(2c-1)} \binom{em}{q}^{q} \left(\frac{2^{2c}}{\sqrt{c}}\right)^{q} \left[\frac{ebM}{(2c-1)q}\right]^{(2c-1)q/b} \left[\frac{(2c-1)q}{bM}\right]^{cq}$$

$$= \sum_{q \leq n/(2c-1)} \left\{ e^{1+(2c-1)/b} b^{(2c-1)/b-c} \left(\frac{2^{2c}}{\sqrt{c}}\right) \times q^{c-(2c-1)/b-1} m \left[\frac{(2c-1)}{M}\right]^{c-(2c-1)/b} \right\}^{q}.$$

Assuming $c > (b-1)/(b-2)$,

$$f < \sum_{q \leq n/(2c-1)} \left\{ e^{1+(2c-1)/b} b^{(2c-1)/b-c} \left(\frac{2^{2c}}{\sqrt{c}}\right) \left(\frac{n}{(2c-1)}\right)^{c-(2c-1)/b-1} \times m \left[\frac{(2c-1)}{M}\right]^{c-(2c-1)/b} \right\}^{q}$$

$$< \left(\frac{n}{(2c-1)}\right) \left\{ e^{1+(2c-1)/b} b^{(2c-1)/b-c} \left(\frac{2^{2c}}{\sqrt{c}}\right) \times \left(\frac{n}{M}\right)^{c-(2c-1)/b-1} m \left[\frac{(2c-1)}{M}\right] \right\}$$

$$= e^{1+(2c-1)/b} b^{(2c-1)/b-c} \left(\frac{2^{2c}}{\sqrt{c}}\right) \left(\frac{n}{M}\right)^{c-(2c-1)/b} m$$

$$= e^{1+(2c-1)/b} b^{(2c-1)/b-c} \left(\frac{2^{2c}}{\sqrt{c}}\right) n^{k-\varepsilon[c-(2c-1)/b]}.$$

Choosing constants $b > 2$ and $c > (bk-\varepsilon)/(\varepsilon(b-2)) > (b-1)/(b-2)$ yields $f < n^{-O(1)}$ since $e^{1+(2c-1)/b} b^{(2c-1)/b-c} (2^{2c}/\sqrt{c})$ is then merely another constant. ∎

Applying this lemma to the algorithm in (Upfal and Wigderson, 1987), we immediately obtain the following theorem.

THEOREM 2. *An arbitrary step of an n-processor P-RAM with memory size* $m = n^{k}$ $(k > 1)$ *can be simulated on a DMMPC with n processors,* $M = n^{1+\varepsilon}$ *memory modules, and redundancy* $r = O((k-\varepsilon)/\varepsilon) = O(1)$ *in time* $O(\log n)$.

3. SIMULATING A P-RAM ON A 2DMOT

The previous section discussed the effect of memory granularity on the redundancy of P-RAM simulations by DMMPCs. It is natural to wonder

what effect memory granularity has on P-RAM simulations by bounded-degree networks. Strictly speaking, however, the BDN model permits only $M = n$ memory modules, each associated with a processor. For this reason, we propose a *distributed-memory, bounded-degree network* (DMBDN) model of computation. In this model, the n RAM processors are interconnected to each other and to $M = rm/g$ memory modules by a bounded-degree network (Fig. 6). Further departing from the restrictions of the BDN model, we also allow the bounded-degree interconnection network to introduce $O(m)$ additional processors, but these are only switches; they need not have any computational power. Note that the MPC model and the BDN model conceal the existence of $m - n$ similar nodes in the address decoding circuitry of the memory modules.

As mentioned earlier, the 2DMOT simulation scheme of (Luccio *et al.*, 1990) introduced $O(n^2)$ additional switching processors. This resulted in a constructive DMBDN with reasonable constants, but the redundancy remained $O(\log n)$ because the memory granularity was unchanged. One approach to reducing memory granularity with a DMBDN would be to implement the algorithm of Section 2 using an $n \times M$ 2DMOT as a crossbar switch between processors and memory modules (Fig. 7). A direct implementation results in an $O(\log^2 n)$-time algorithm with redundancy $r = O((k - \varepsilon)/\varepsilon)$, while using $O(nM)$ additional switches. The pipelining strategies of (Luccio *et al.*, 1990) can reduce the time complexity to $O(\log^2 n/\log\log n)$. As an extreme case, consider $M = m$. Obviously, redundancy $r = 1$ suffices to achieve $O(\log n)$ time in this case.

Another 2DMOT simulation scheme deploys the M modules at the leaves of the 2DMOT and the n processors at the roots of the first n row trees, provided $M = O(n^{2+\delta})$ (for simplicity, we identify row and column tree roots, Fig. 8). This simulation scheme, which is admitted by the DMBDN model, introduces only $O(n + M) = O(M)$ additional switches, but still reduces memory granularity and can thereby achieve constant redundancy.

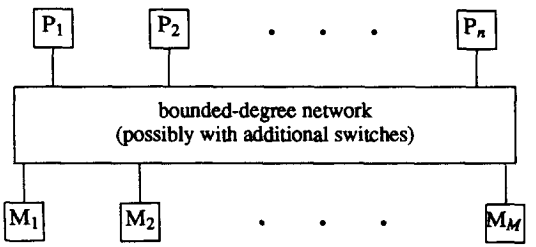


FIG. 6. The DMBDN model of computation.

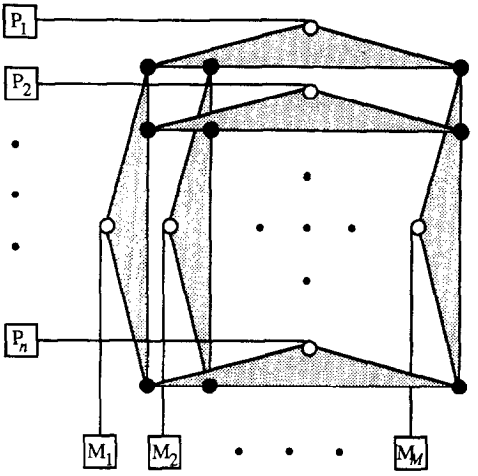


FIG. 7. The 2DMOT for constant redundancy P-RAM simulation with $O(nM)$ additional switches (shaded triangles represent balanced binary row and column trees).

THEOREM 3. *If m is polynomial in n and $M = n^{2+\delta}$ for constant $\delta > 0$, then a $\sqrt{M} \times \sqrt{M}$ 2DMOT can deterministically simulate a P-RAM step in $O(\log^2 n / \log\log n)$ time with redundancy $r = O(1)$.*

Proof. The simulation scheme works in essentially the same manner as that in (Luccio *et al.*, 1990), except for the routing of access requests. When

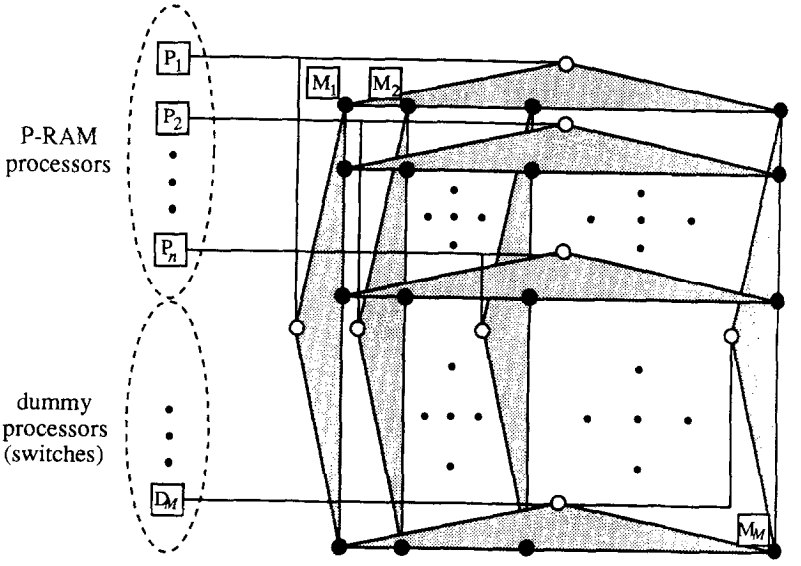


FIG. 8. The 2DMOT for constant redundancy P-RAM simulation with $O(M)$ additional switches.

processor P_l must access a variable copy stored in the memory module $M_{i,j}$ located in row i and column j, it sends the request down the lth row tree to the jth leaf. From there, it propagates up to the root of the jth column tree (provided it does not collide with a conflicting request), whence it is sent down to the ith leaf, i.e., $M_{i,j}$. The answered request returns to P_l simply by reversing this path. Other than this, the simulation proceeds as in (Luccio *et al.*, 1990), with processors organized as clusters cooperating to retrieve copies of the variables needed by one another. However, since there are now effectively $M' = \sqrt{M} = n^{1+\delta/2}$ memory modules (the $\sqrt{M}$ distinct columns), Lemma 2 can be applied to obtain $O(1)$ redundancy. (In fact, we can simultaneously access along both rows and columns, which further reduces the redundancy by a factor of 2, as can be shown by a modification of Lemma 2.) The time complexity remains $O(\log^2 n/\log\log n)$ as in (Luccio *et al.*, 1990). ∎

As indicated above, the key property of the 2DMOT that we are exploiting here is the fact that a $\sqrt{M} \times \sqrt{M}$ 2DMOT provides us with bandwidth $O(\sqrt{M})$ for memory access. In contrast, each memory module in an MPC or BDN has bandwidth 1, despite the fact that they would require area $O(m/n)$ and perimeter $O(\sqrt{m/n})$ in VLSI. The 2DMOT simply makes better use of the available perimeter. Furthermore, if $g = \Omega(\log^2 n)$, then the 2DMOT P-RAM simulator can be laid out in $O(m)$ area in VLSI, which is clearly optimal. It is also well suited to multi-chip implementations since the required interchip connections can all be made on the perimeters of the chips.

4. Conclusions and Open Problems

We have proposed a feasible 2DMOT architecture that performs general-purpose computations by simulating a P-RAM. By introducing the DMBDN model, we eliminated an unnecessarily severe restriction on the memory bandwidth of parallel computers and, thus, reduced the memory redundancy required for deterministic P-RAM simulation. Although we have removed these restrictions, our 2DMOT architecture still appears to be well-suited to VLSI or multichip implementation.

The DMBDN model gives the designer of general-purpose parallel computers freedom that might be useful in other ways. For instance, the increased memory bandwidth may make it possible to rid deterministic P-RAM simulation schemes of the nonconstructive memory map that must be stored in each processor. A memory map that could be constructed by simple computations within a processor would eliminate the large ($O(m \log rm)$ bits) address look-up table that each processor must store.

Failing in this, it may still be possible to simulate a P-ROM, a parallel, read-only memory, that would support simultaneous address look-up for all processors, and, thus reduce the total look-up table size from $O(mn \log rm)$ to $O(rm \log rm)$ bits.

The derivation of lower bounds in this new model also poses some interesting questions. For instance, the arguments used in (Alt *et al.*, 1987; Karlin and Upfal, 1986) to prove the $O(\log^2 n/\log \log n)$ deterministic time lower bound no longer apply in the DMBDN model. Therefore, it may be possible to speed up these DMBDN simulations. It would be interesting to derive a coresponding nontrivial lower bound on the time complexity of a DMBDN simulation of a P-RAM, especially if the point-to-point communcation restriction can be removed.

RECEIVED December 23, 1988; FINAL MANUSCRIPT RECEIVED December 11, 1989

REFERENCES

ALT, H., HAGERUP, T., MEHLHORN, K., AND PREPARATA, F. P. (1987), Deterministic simulation of idealized parallel computers on more realistic ones, *SIAM J. Comput.* **16**, 808.

FORTUNE, S., AND WYLLIE, J. (1978), Parallelism in random access machines, *in* "Proceedings of the 10th Annual ACM Symposium on the Theory of Computing, San Diego, CA, May 1978," pp. 114–118.

GIFFORD, D. K. (1979), Weighted voting for replicated data, *in* "Proceedings of the 7th Annual ACM Symposium on Operating System Principles, Pacific Grove, CA, Dec. 1979," pp. 150–159.

HERLEY, K. T., BILARDI, G. (1988), Deterministic simulations of PRAMs on bounded degree networks, *in* "Proceedings of the 26th Annual Allerton Conference on Communication, Control and Computing, Monticello, IL, Oct. 1988," pp. 1084–1093.

HIRSCHBERG, D. S. (1977), "Fast Parallel Sorting Algorithms," Technical Report, Department of Electrical Engineering, Rice University, Houston, TX.

KARLIN, A. R., AND UPFAL, E. (1980), Parallel hashing—An efficient implementation of shared memory, *in* "Proceedings of the 18th Annual ACM Symposium on the Theory of Computing, Berkeley, CA, May 1986," pp. 160–168.

KARP, R. M., AND RAMACHANDRAN, V. (1988), "A Survey of Parallel Algorithms for Shared-Memory Machines," Technical Report UCB/CSD 88/408, Computer Science Department, University of California at Berkeley, Berkeley, CA.

LEIGHTON, F. T. (1984), New lower bound techniques for VLSI, *Math. Systems Theory* **17**, 47.

LUCCIO, F., PIETRACAPRINA, A., AND PUCCI, G. (1988), A probabilistic simulation of PRAMs on a bounded-degree network, *Inform. Process. Lett.* **28**, 141.

LUCCIO, F., PIETRACAPRINA, A., AND PUCCI, G. (1990), A new scheme for the deterministic simulation of PRAMs in VLSI, *Algorithmica* **5**, 529.

MEHLHORN, K., AND VISHKIN, U. (1984)W, Randomized and deterministic simulations of PRAMs by parallel machines with restricted granularity of parallel memories, *Acta Informat.* **21**, 339.

NATH, D., MAHESHWARI, S. N., AND BHATT, P. C. P., Efficient VLSI networks for parallel processing based on orthogonal trees, *IEEE Trans. Comput.* **C-32**, 569.

PREPARATA, F. P. (1977), Parallelism in sorting, *in* "Proceedings of the 1977 International Conference on Parallel Procesisng, St. Charles, IL, Aug. 1977," pp. 202–206.

RABIN, M. O. (1989), Efficient dispersal of information for security, load balancing, and fault tolerance, *J. Assoc. Comput. Mach.* **36**, 335.

RANADE, A. G. (1987), How to emulate shared memory, *in* "Proceedings of the 28th Annual Symposium on the Foundations of Computer Science, Los Angeles, CA, Oct. 1987," pp. 185–194.

SCHUSTER, A. (1987), How to share memory in a distributed system using a small extra space, unpublished manuscript, Computer Science Department, Hebrew University, Jerusalem.

THOMAS, R. H. (1979), A majority consensus approach to concurrency control for multiple copy databases, *ACM Trans. Database Systems*, Jun. 1979, 180.

UPFAL, E. (1984), A probabilistic relation between desirable and feasible models of parallel computation, *in* "Proceedings of the 16th Annual ACM Symposium on the Theory of Computing, Washington, D.C., May 1984," pp. 258–265.

UPFAL, E., AND WIGDERSON, A. (1987), How to share memory in a distributed system, *J. Assoc. Comput. Mach.* **34**, 116.